Report of MILC Layout Project

Jun He, Jim Kowalkowski, Marc Paterno, Don Holmgren, Jim Simone

August 24, 2011

Abstract

Different layouts lead to different inter-node/intra-node communication cost. This project explores how the layout of subvolumes influences the performance of MILC. In this report we describe the layout supports we added to MILC, prove the impacts of these factors by experiments and give suggestions on how to arrange MILC subvolumes.

Contents

Part I

Dictionary

Chapter 1

Dictionary

1. Lattice size: The size of the lattice defined in the MILC input file by nx,xy,xz and nt. Such as $12 \times 12 \times 12 \times 12$.

2. subvolume: a part of the lattice that can be handled by one core. It is measured by site. For example, a subvolume size is $6 \times 3 \times 3 \times 3$, which means this subvolume has 6 sites in X dimension and 3 sites in the others.

3. nodevolume: the subvolumes a node has. If a nodevolume is $A \times B \times C \times D$, it has A subvolumes in X dimension, B subvolumes in Y dimension, ... it has A subvolumes in x dimension, B subvolumes in Y dimension, ... it has A subvolumes in x dimension, B subvolumes in Y dimension, ...

4. Nsquares: the number of subvolumes in each dimension. For example, if the lattice size is $12 \times 12 \times 12 \times 12$ and the subvolume size is 6x3x3x3, then nsquare is 2x4x4x4.

5. NodeCut: how a lattice is cut into nodevolumes. For example, a NodeCut of $1 \times 1 \times 2 \times 2$ cuts lattice $12 \times 12 \times 12 \times 12$ into 4 nodevolumes, each of them has $12 \times 12 \times 6 \times 6$ sites. If the Nsquares is $4 \times 4 \times 4 \times 2$, then each node has NodeVolume of size $4 \times 4 \times 2 \times 1$.

6. TPI: Time Per congrad5 Iteration.

Part II

Introduction

MILC is a old but still very active project. It can take thousands of cores and run for days or even months. It is very important to accelerate it so that scientists can get the results as soon as possible.

Nowadays we have more cores and nodes, which means more computing power. However, more separated computing units lead to more communication. The cost of communication leads to great degradation of performance. So reducing the communication cost may improve the overall performance a lot.

There are several elements in MILC, including lattice, subvolume, site. 1.1 shows their relationships. The original lattice is comprised of many sites in four dimensions (x, y, z, t). It is cut to many subvolumes. Each of them occupies a core. Each of these subvolumes has many sites in it. Each subvolume exchanges data with its adjacent subvolumes. Communication in MILC is a important factor. Since some communication paths are slow and some are fast, it is crucial to layout the subvolumes properly so that less communication resides on slow paths.

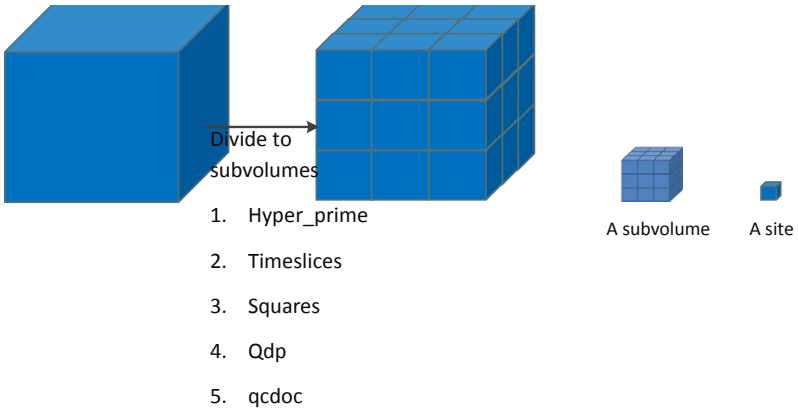

Figure 1.1: How MILC works

In this project, we first added our own layout support to MILC and then explored many factors that have impacts on the performance.

Part III

Design

Chapter 2

Adding layout support

2.1 NodeVolume

The way that MILC maps subvolumes to cores is to assign each subvolume a rank by its coordinates. In order to map subvolumes to cores in the manner that we want, we introduced a new remapping layer by which we can assign a particular group of subvolumes to a node (in this report, node always refers to a node of a computer cluster).

In order to describe the group of subvolumes that are assigned to the same node, we introduced the concept of NodeVolume. NodeVolume can be described by 4-d notation like $A \times B \times C \times D$, which means in this node there are A consecutive subvolumes in X dimension, B in Y dimension, C in Z dimension, and D in T dimension. For example, in fig. 2.1, we demonstrate what a NodeVolume is. We modified MILC to take NodeVolume as input from the input file so that we can simply change the numbers in input file to control the layout.

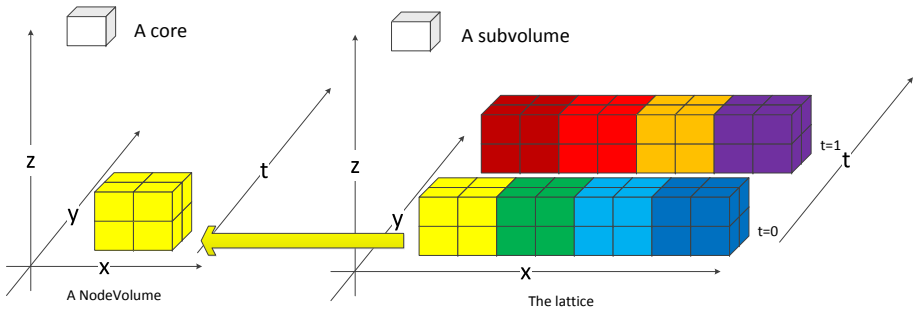

Figure 2.1: A demostration of NodeVolume. Size of lattice: 8x2x2x2, number of cores per node: 8. NodeVolume is 2x2x2x1. Subvolumes with same color are on the same node.

2.2 Printing out layout information

In order to layout the subvolumes correctly, we need to check if it is put in the right place. So we added layout information to output. The CoreMap, NUMA node number and lastcpu No. were got by Hwloc, a project used to bind processes to cores in multi-core environment. The lines below are the sample output. The output can be further parsed and read by R for analyses.

> I am rank [114] out of [256] on machine [ds2004]. PID [31907]. Subvolume Coordinate (0, 3, 2, 4). CoreMap: 0x00040000 numa 4, lastCPU: 0x00040000 18
> I am rank [116] out of [256] on machine [ds2004]. PID [31914]. Subvolume Coordinate (0, 3, 0, 5). CoreMap: 0x00100000 numa 5, lastCPU: 0x00100000 20
> I am rank [169] out of [256] on machine [ds2006]. PID [1526]. Subvolume Coordinate (0, 1, 5, 2). CoreMap: 0x00000200 numa 2, lastCPU: 0x00000200 9
> ...

2.3 Mapping framework of MILC

There are several function calls in MILC that manage the mapping from sites to nodes, or from nodes to sites. In order to insert a remapping, we modified or replaced these functions. In this project, we assume that MPI assigns ranks to nodes in the fashion like fig. 2.2, in which MPI fills up the cores of one node before filling the next node. It is the default setting of most MPI implementation. If not, we can control it by options like:

> -bynode, - -bynode
> Allocate (map) the processes by node in a round-robin scheme.
> -byslot, - -byslot
> Allocate (map) the processes by slot in a round-robin scheme. This is the default. –byslot option tells Open MPI to use all slots on an available node before allocating resources on the next available node.

-byslot should be the option.

2.4 MPI trace support added to MILC

In order to get the traces of the communication in MILC, we instrumented the MPI calls to record the parameters. By doing so, we are able to record the source, destination, size and etc. of every message. The format of the traces is like:

> 12 12 12 12 2 4 4 4 1 2 4 4 0 MPI_Barrier X X X X X
> 12 12 12 12 2 4 4 4 1 2 4 4 0 MPI_Bcast 1568 4 0 X 91

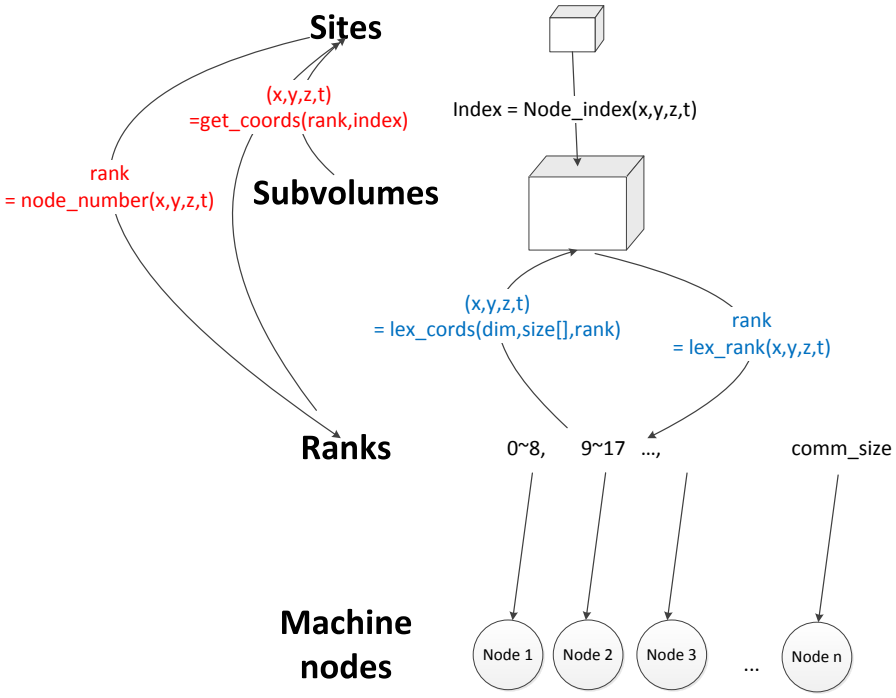

Figure 2.2: Mapping Framework of MILC.

12 12 12 12 2 4 4 4 1 2 4 4 0 MPI_Isend 1 4 32 0 91
12 12 12 12 2 4 4 4 1 2 4 4 0 MPI_Irecv 1 4 32 0 91

The columns are LatticeX LatticeY LatticeZ LatticeT Nsquares_X Nsquares_Y Nsquares_Z Nsquares_T Nodevolume_X Nodevolume_Y Nodevolume_Z Nodevolume_T MyRank Call Count TypeSize End Tag Comm. We can read the traces to R for analyse. Although there are millions of lines for one run, R can do vector computation quite fast.

Part IV

Explorations

Chapter 3

Experiment Setup

3.1 Environment

All of the experiments in this report were conducted in D/S cluster. DS is a 245-node cluster with quad-socket eight-core Opteron 6128 (2.0 GHz) processors and a quad-data-rate Infiniband fabric. Two seperated groups of nodes are reserved for the experiments. One group has 4 nodes (DS1613-1616). One group has 8 nodes (DS2001-DS2008). The nodes within the same group are connected to the same Infiniband switch. So the inter-node connections are identical between any two nodes of the same group.

We use the application ks_imp_dyn in MILC directory for evaluation. It was compiled and ran by mvapich-1.2rc1 via NUMA binding wrapper numa_32_mv2. One exception was that we used Openmpi with Tau for instrumenting the MILC. The version of Tau was 2.20.2 with pdtoolkt 3.16.

3.2 Metrics

In our experiment, we use TPI to evaluate the performance. TPI stands for Time Per Iteration of each CONGRAD5 step. For example, for the following CONGRAD5 step, the TPI is $8.044958 \times 10^{-03}/19 = 4.234188 \times 10^{-4}$.

> CONGRAD5: time = 8.044958e-03 (fn F) masses = 1 iters = 19 mflops = 4.541461e+02

In the graph presented in this report, we only plotted the median TPI of all CONGRAD5 steps for the same layout. By doing so the actual and predicted TPI can be easily compared.

Each layout has been run for over 5 times (or even more) with over hundreds of iterations in order to get enough samples for statistics.

3.3 Input file

A typical input file is like this:

```
prompt 0
nflavors1 2
nflavors2 1
nx 16
ny 16
nz 16
nt 16
gx 1
gy 2
gz 4
gt 4
node_geometry 2 4 4 4
iseed 5682304
warms 0
trajecs 1
traj_between_meas 1
beta 6.85
mass1 0.234e-5
mass2 0.412e-5
u0 0.8441
microcanonical_time_step 0.01
steps_per_trajectory 100
max_cg_iterations 1000
max_cg_restarts 5
error_per_site 0.2377e-3
error_for_propagator 0.1865e-3
fresh
forget
```

gx, gy, gz and gt are the four parameters we added for controlling NodeVolume. Sometimes, we change steps_per_trajectory to tune the overall running time of a single run. The other parameters remained the same in all the experiments.

3.4 Linear Model

In this report, we use R to analyze performance results. First, the data was pulled from the output file of MILC by a Python script. Then we fed the formated data to R for analyse. We used lm() in R to get the linear model for TPI and then predicted TPI by predict() based on the linear model. A typical *lm()* call is like *lm(tpi.median var1+ var2)*.

Chapter 4

Communication Vs. Computation

The main assumption of this project is that the communication cost is big in MILC and it is worth reducing it for better performance of MILC. It is also worth knowing that how the size of lattice influences the communication cost. Here we analyse the proportion of MPI_Wait() in the whole time of the MILC. Based one the process of MPI_Isend()/MPI_Irecv() -> computation -> MPI_Wait(), longer time spent on MPI_Wait() means larger communication cost.

We instrumented MILC by Tau and ran it with different lattice sizes. The results are presented in fig. 4.1 and 4.2. These two runs are all the same except that they have different lattice sizes. We can see that $48 \times 48 \times 48 \times 48$ spent more time on communication (MPI_Wait(), left blue bars) than $16 \times 16 \times 16 \times 16$. In addition, when lattice size goes up, the communication time becomes unstable (large standard deviation).

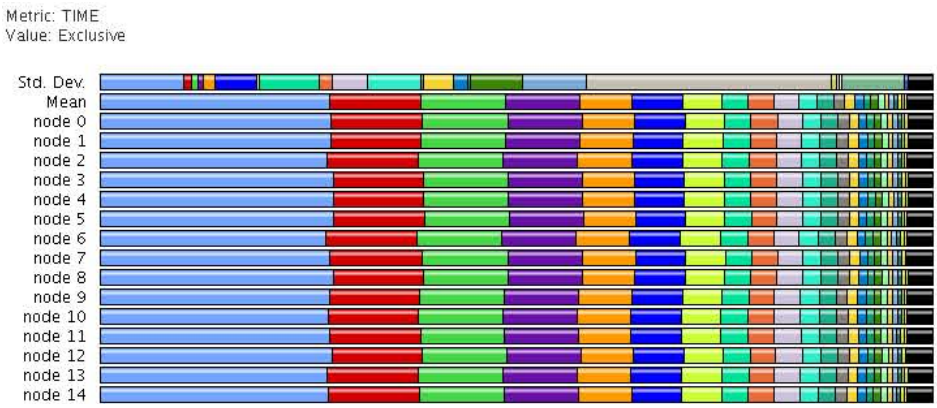

Figure 4.1: 16x16x16x16, 4 node. Proportion of each function. Blue bars on the left is MPI_Wait()

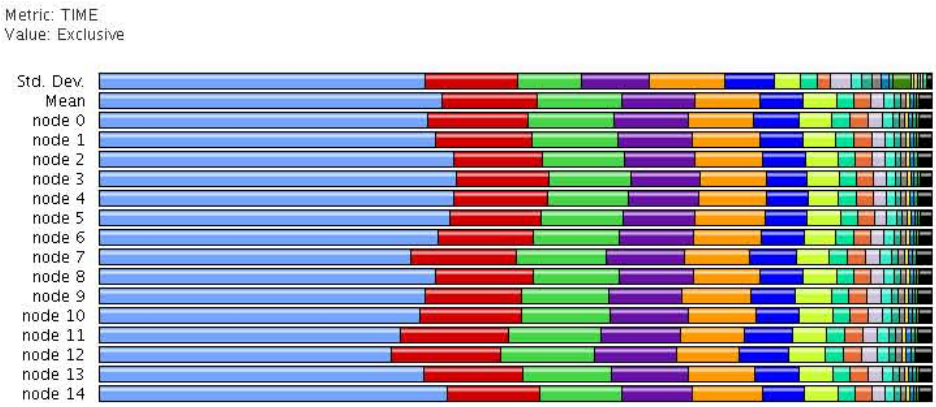

Figure 4.2: 48x48x48x48, 4 node. Proportion of each function. Blue bars on the left is MPI_Wait()

Fig. 4.3 and 4.4 are the communication heat maps for lattice $16 \times 16 \times 16 \times 16$ and $48 \times 48 \times 48 \times 48$. We can see that they have exactly the same pattern for number of communication calls and total volume bytes. The difference is that $48 \times 48 \times 48 \times 48$ has slightly more calls than $16 \times 16 \times 16 \times 16$, but has much more (two magnitude) data.

The guess is that when the lattice size goes up (everything else remains the same, e.g. number of nodes, layout of subvolumes), the size of data transferred goes up a lot (proportional to the increase of the size of the lattice). But the number of communnication calls just changes a little bit (because the number of subvolumes remains the same). The portion of communication time over total time grows. To sum up, with bigger lattice size, it is more communication dominant.

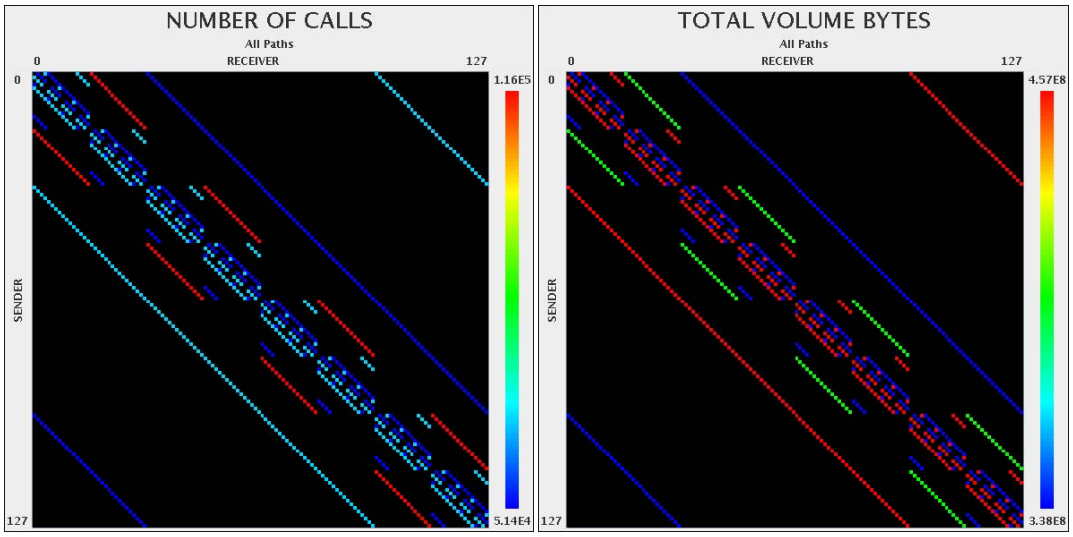

Figure 4.3: Communication heat map, 16x16x16x16, 4 node.

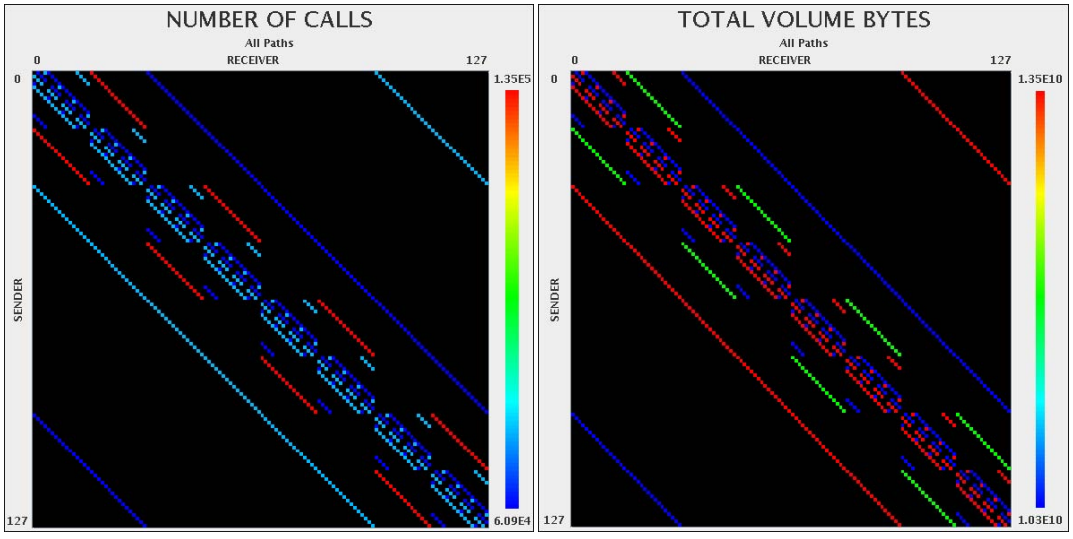

Figure 4.4: Communication heat map, 48x48x48x48, 4 node.

Chapter 5

Inter-node Subvolume Path

The definition of Inter-node Subvolume Path in dim Q is the number of pairs of subvolumes exchanging data in dim Q. This parameter affects the number of packages sending over dim Q. And the number of packages has great impact on the performance of Infiniband, since each package has individual startup time.

The total ISP is determined by Nsquares and NodeCut (refer to chapter 1 for definition of Nsquares and NodeCut). If Nsquares is $A \times B \times C \times D$, NodeCut is $Q \times R \times S \times T$ ($Q = 1$, $R = 1$, $S > 1$, $T > 1$), the overall ISP in this case is $A \times B \times T \times S + A \times B \times C \times T$. $Q = 1$ and $R = 1$ mean that X and Y dimensions are not cut and there is no inter-node path in these dimensions. For example, if Nsquares is $4 \times 4 \times 2 \times 1$, NodeCut is $1 \times 1 \times 2 \times 2$, ISP in Z dimension is 4*4*1=16 in one inter-node slot. The ISP in T dimension is $4 * 4 * 2 = 32$. In total, ISP is $16 * 2 + 32 * 2 = 96$.

For the cases with the same number of subvolumes, the numbers of communication paths between subvolumes are the same. However, the number of inter-node paths can be different because the subvolumes can be assigned to the nodes in different ways.

Different numbers of ISPs lead to different number of MPI_Isend() between nodes.(MILC uses MPI_Isend() for communication between subvolumes.) Below is a table including the ISP in theory and in practice (data was from the traces we got).

Table 5.1: ISP in thoery and MPI_Isend in practice

Nsquares	$2 \times 4 \times 4 \times 4$	$2 \times 4 \times 4 \times 4$
NodeCut	$1 \times 1 \times 1 \times 4$	$2 \times 1 \times 1 \times 2$
ISP in theory for one time of data exchange	$(2 \times 4 \times 4) \times 4 = 128$	$(2 \times 4 \times 4) \times 4 + (2 \times 4 \times 4) \times 2 = 192$
Number of Inter-node MPI_Isend() calls in practice	471936	706688

$471936/706688 = 0.6678$, $128/192 = 0.6667$ The ratios in theory and in practice are constant, which indicates that number of ISP is proprotional to the number of MPI_Isend() in practice and it reflects the number of communication packages.

5.1 Experiments

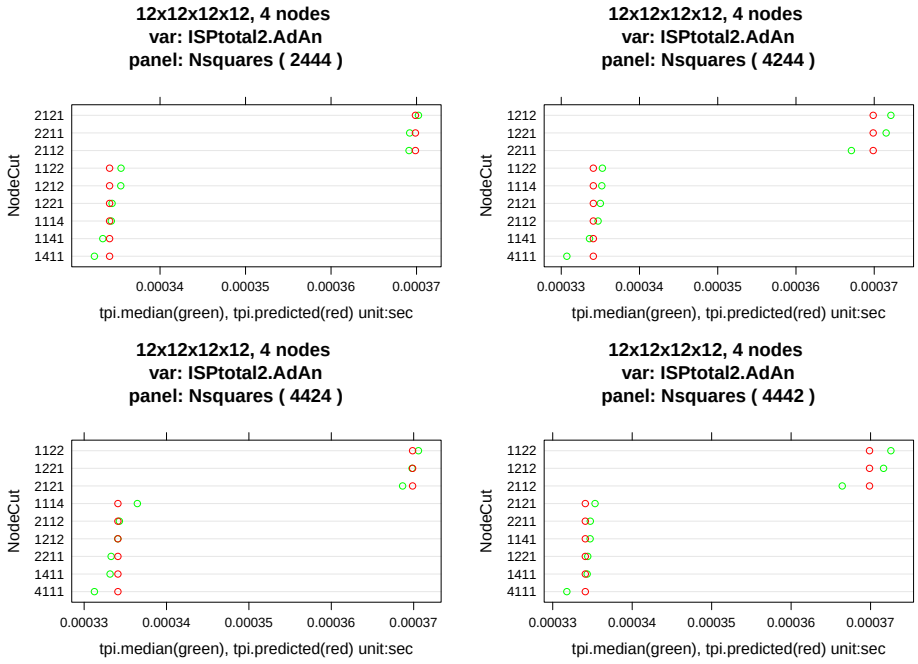

Figure 5.1: $12 \times 12 \times 12 \times 12$, 4 nodes. Due to the limited space, NodeCut is presented in a shorter version. For example, 2121 means $2 \times 1 \times 2 \times 1$.

ISPtotal2.AdAn is the number of ISP of all dimentions and all nodes. It is same as the ISP mentioned above, but with a different notation (Here AdAn means it is in All dimension All nodes. We have something else about ISP in our experiments so we use a special name for this ISP here.). It is calculated by the R codes in Appendix .1. e is the the data frame that has all the data, including layout information of each run.

We can see in fig. 5.1 that the median of tpi roughly matches the pattern predicted by ISPtotal.AdAn. ISPtotal.AdAn can categorize the performances to two parts. Clearly, there are some other factors affecting the performance. But the ISP should be the dominant one since the tpi.median are widely separated. In this experiment, the number of sites on the inter-node surfaces are all the same, so it is not the factor that affecting the performance.

To further confirm that the total ISP is actually different in real runs, we profiled MILC by TAU. We take NodeCut $1\times1\times2\times2$ and $4\times1\times1\times1$ of Nsquares $4\times4\times4\times2$

as an example (lattice size is $12 \times 12 \times 12 \times 12$). The result is shown in fig. 5.2 and 5.3. The X and Y axes are the rank numbers, indicating receivers and senders. Because we have 4 D/S nodes in this case, we have 128 ranks. Rank 0-31 are in Node 0. Rank 32-63 are in Node 1. Rank 64-95 are in Node 2. Rank 96-127 are in Node 3. The colors indicate the number of calls on a particular path. Red indicates the most.

In fig. 5.2, we can see that the red paths are all within one node. But in 5.3, we can see all the red paths are inter-node. In theory, the number of paths on the inter-node surface is $4 * 4 * 4 = 64$ in T dimension and $4 * 4 * 2 = 32$ in Y dimension. By the number in the graph, we can confirm that the number of paths between node 0 and node 3 (T dimension) is twice that between node 0 and node 1 (Z dimension).

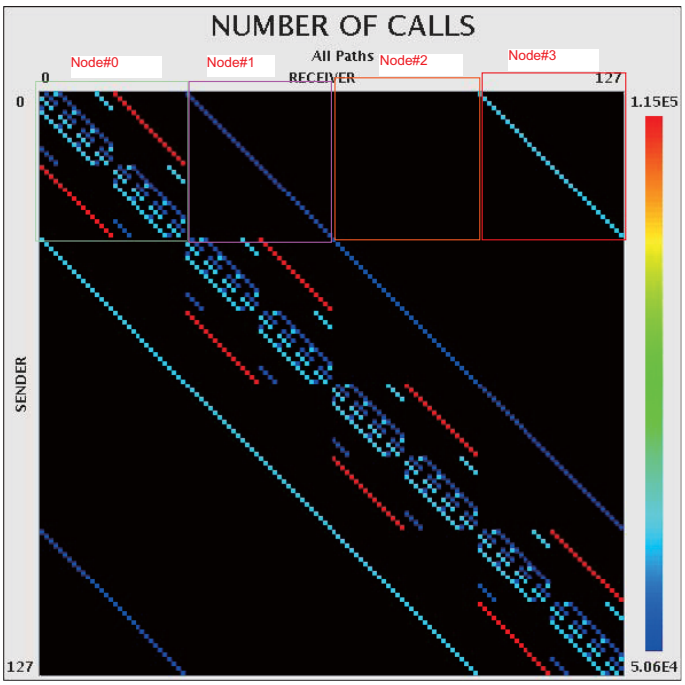

Figure 5.2: $12 \times 12 \times 12 \times 12$, 4 nodes, NSquares: $4 \times 4 \times 4 \times 2$, NodeCut, $4 \times 1 \times 1 \times 1$

In this case of lattice size $16 \times 16 \times 16 \times 16$, ISPtotal2.AdAn can still categorize the performance (5.4). In addition, we can intuitively tell the difference by the the heat graphs (fig. 5.5 and fig. 5.6). NodeCut 1411 has better performance since there are less inter-node calls (or paths).

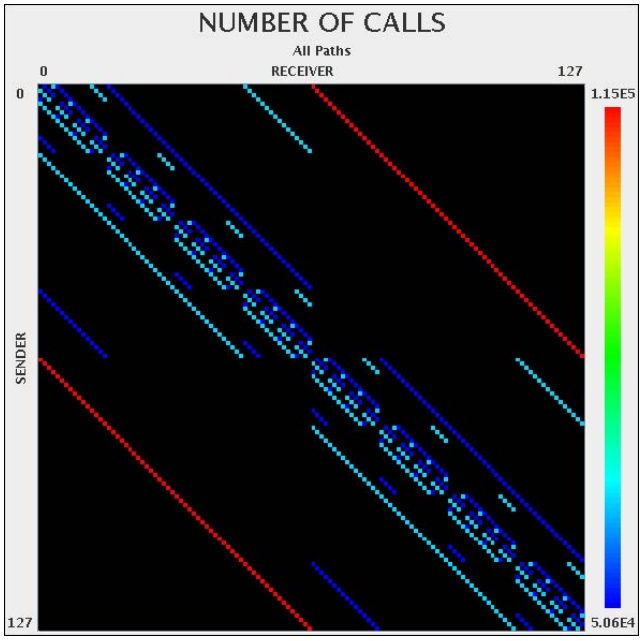

Figure 5.3: $12 \times 12 \times 12 \times 12$, 4 nodes, NSquares: $4 \times 4 \times 4 \times 2$, NodeCut, $1 \times 1 \times 2 \times 2$

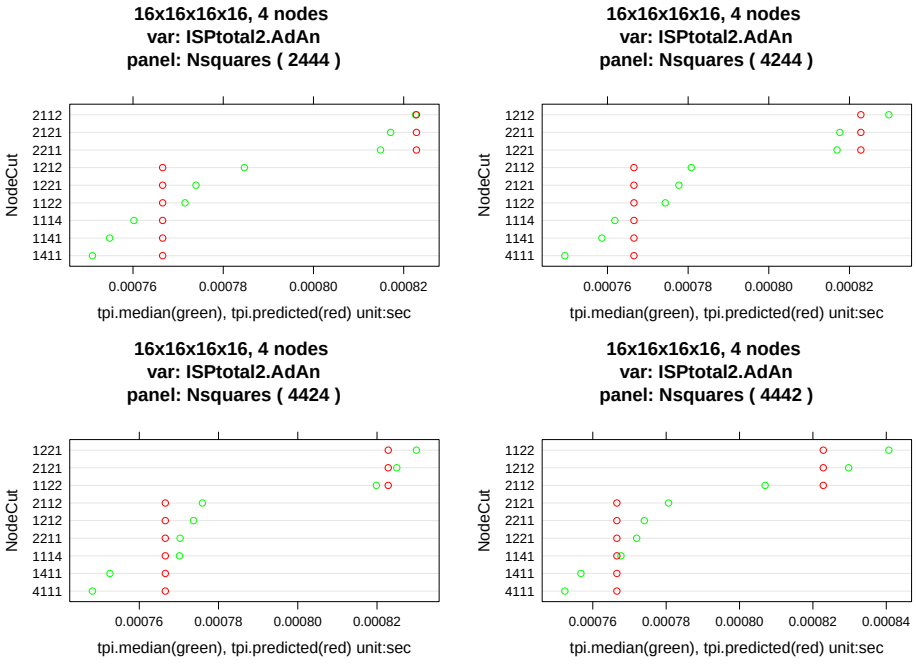

Figure 5.4: 16x16x16x16, 4 nodes. We can predict the performance by ISPtotal2.AdAn

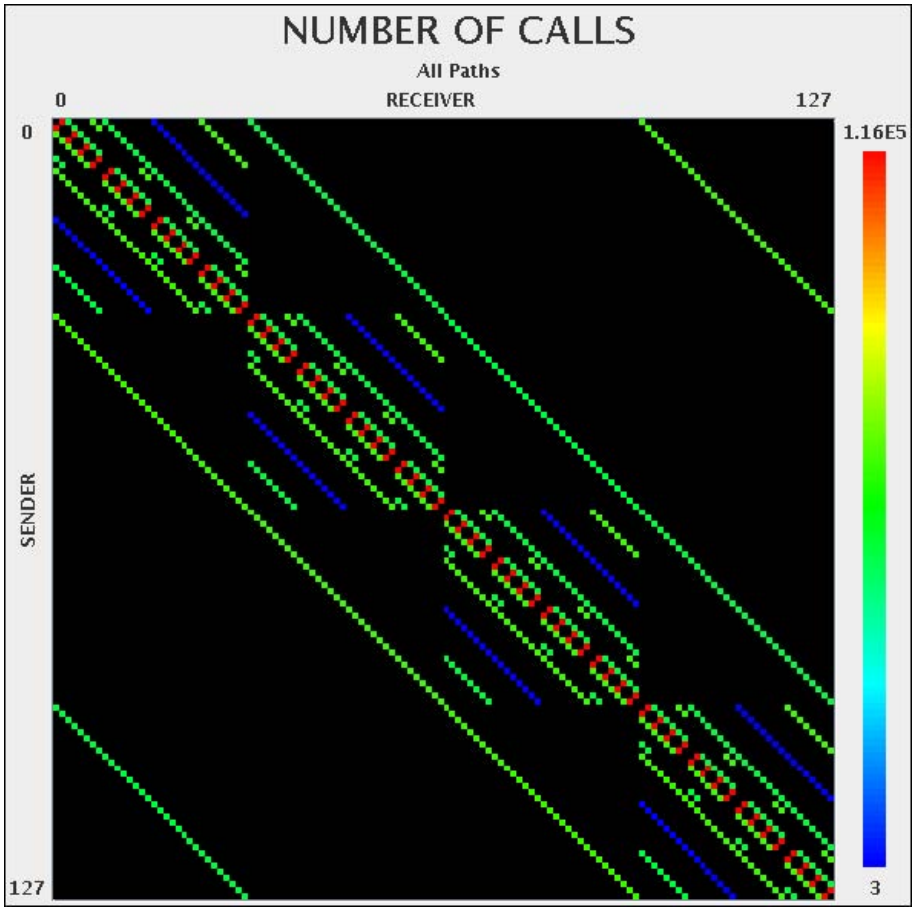

Figure 5.5: 16x16x16x16, 4 nodes, NSquares: 2444, NodeCut, 1411

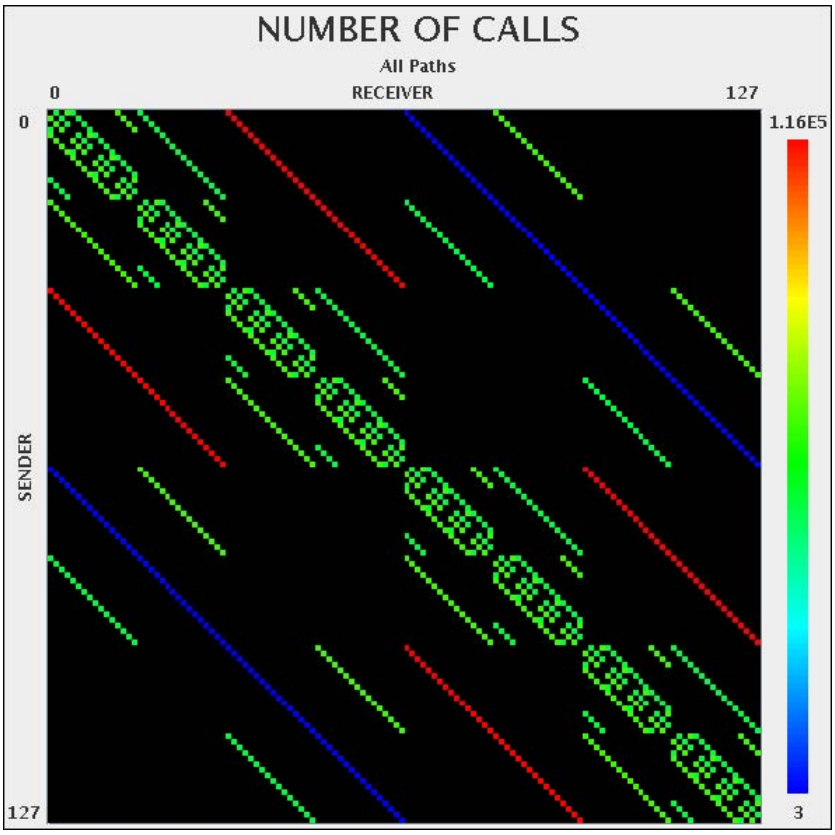

Figure 5.6: 16x16x16x16, 4 nodes, NSquares: 2444, NodeCut, 2112

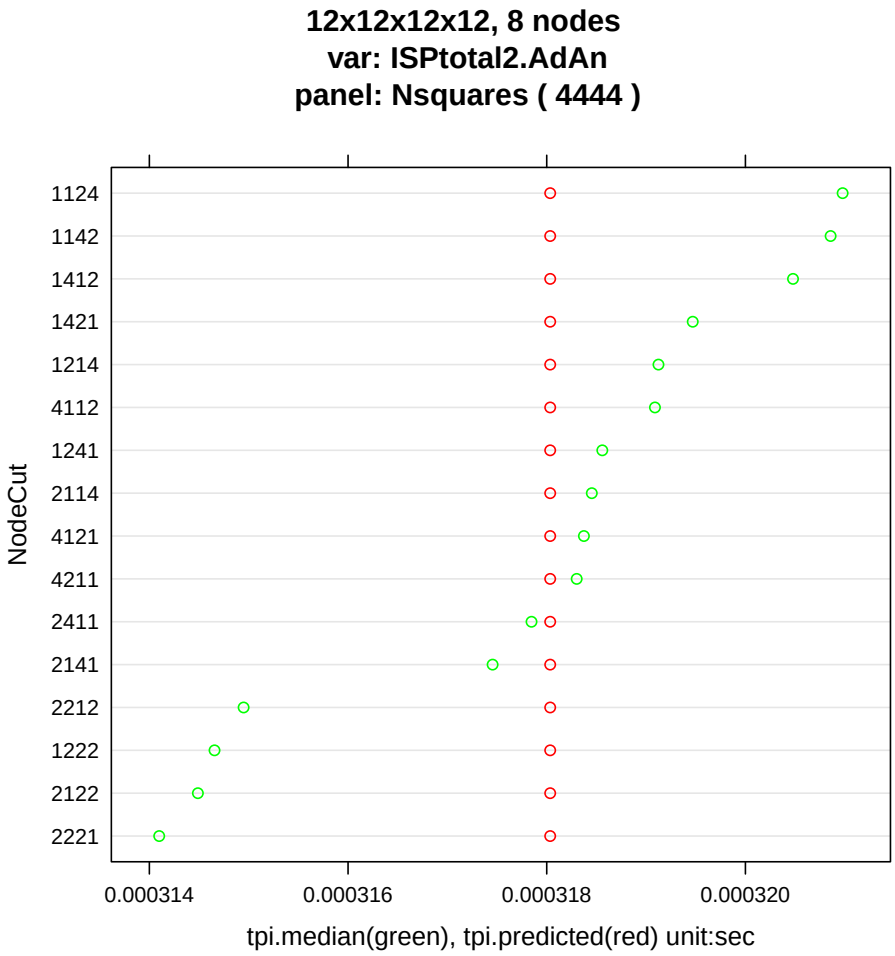

Figure 5.7: $12 \times 12 \times 12 \times 12$, 8 nodes. In this case, ISPtotal2.AdAn is all the same no matter how the lattice is cut. The range of tpi.median is $0.000320 - 0.000314 = 0.000006$. The difference is only around 2%. The performance in this case can be predicted by SITESmax.1nAd (refer to fig. 7.1), ISPmax.Ad1n or other variables related to MAX number of ISP in one of the 4 dimension. All of these variables are proportional to each other, it must be their combined effects led to the difference of performance.

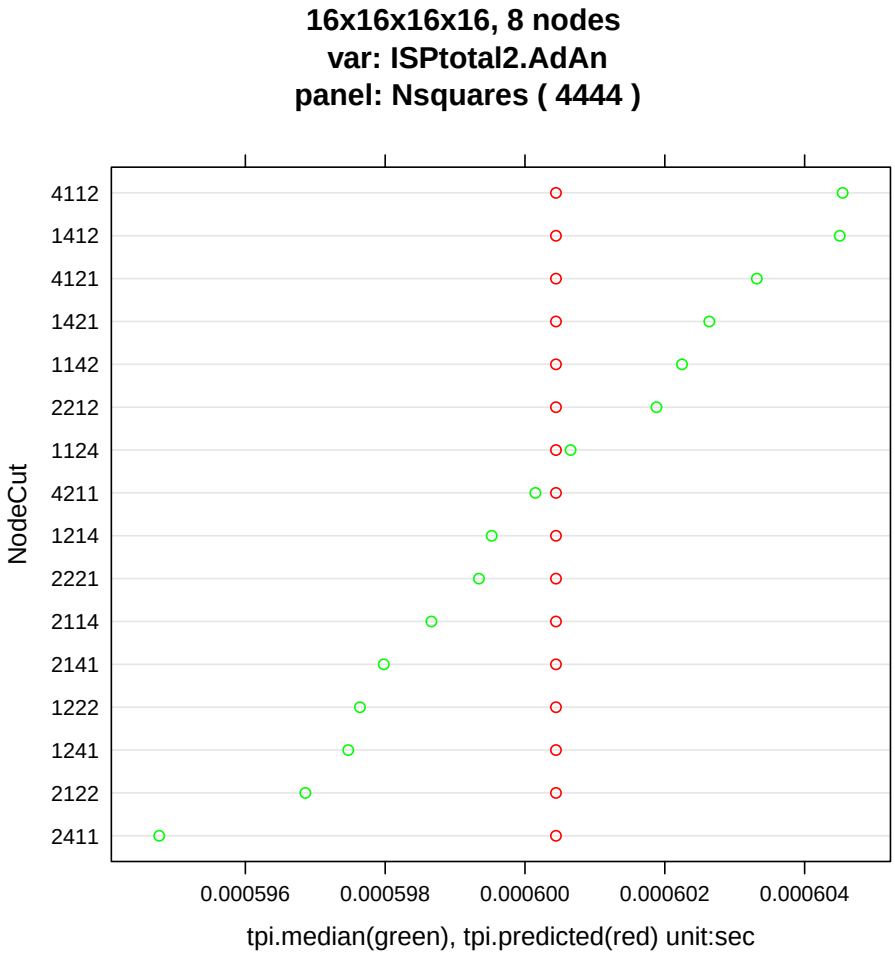

Figure 5.8: $16 \times 16 \times 16 \times 16$, 8 nodes. In this case, none of the variables we have inspected can predict the performance. The range of the tpi.median is 0.000595-0.000604=0.000009. The difference is around 1.5%. The guess is that all the variables are so balanced there is not a dominant factor that can predict the performance. Comparing this with fig. 5.7, they are all the same except for the lattice size. It is indicating that when size goes up, some variables become less dominant and some become more dominant.

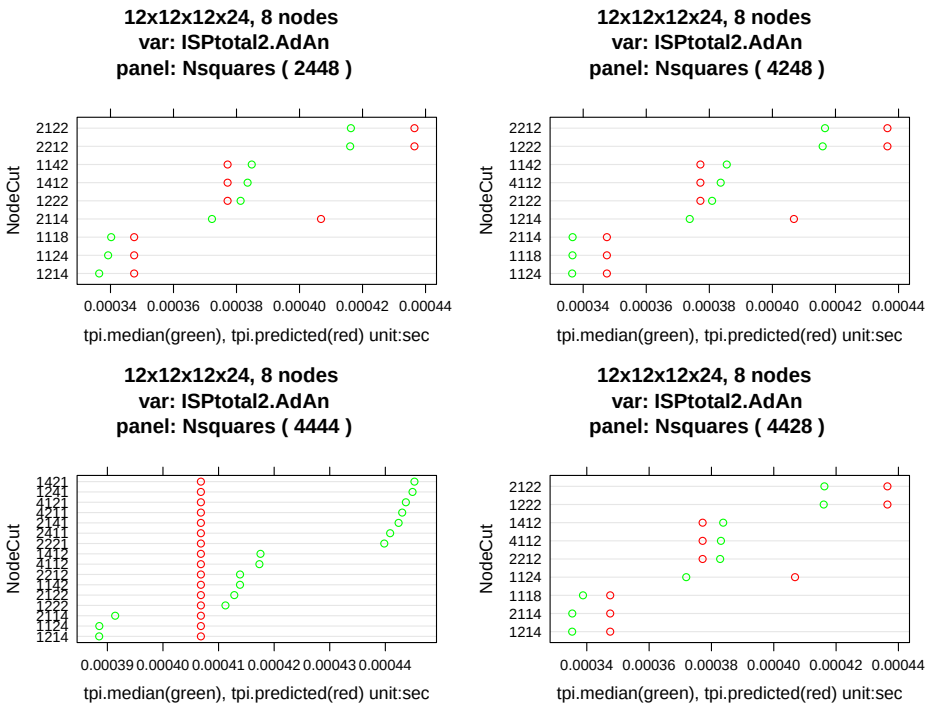

Figure 5.9: $12 \times 12 \times 12 \times 24$, 8 nodes. We can see that the range of tpi.median for Nsquares $4 \times 4 \times 4 \times 4$ is around 0.00039-0.00045 (0.0006), while the ranges of the others are around 0.00034-0.00043(0.0009). When Nsquares is $4 \times 4 \times 4 \times 4$, the ISPtoatal2.AdAn is the same, no matter how you cut the lattice. In this case, the dominant factor is something else (sites on the inter-node surface, refer to 6).

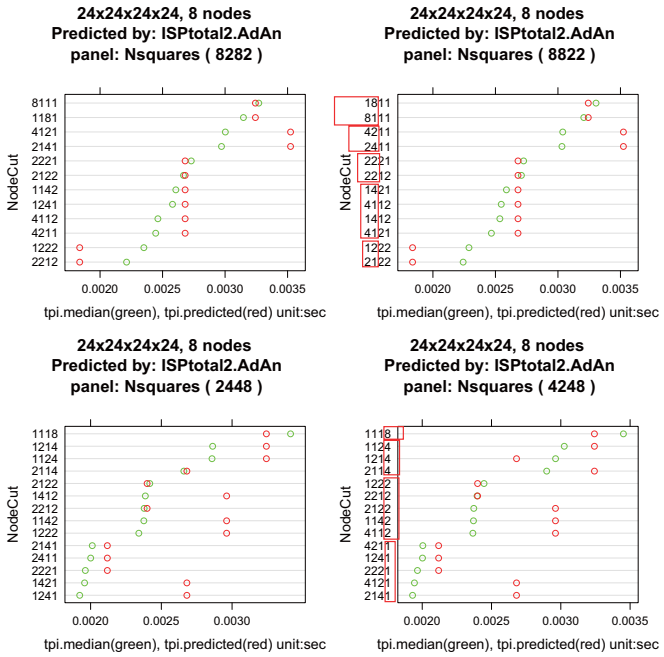

Figure 5.10: $24 \times 24 \times 24 \times 24$, 4 nodes. ISPtotal2.AdAn can not predict tpi.median. But we can easily find the pattern by looking at how the biggest Nsquares dimension(s) is cut. For example, in Nsquares 4248, if T dimension is cut into 8 pieces, it is the slowest. If T dimension is not cut at all, the performance is in the fastest group. In this cast, T dimension has most of the subvolume communication paths (there are 8 subvolumes in T dimension). The guess is that for this case, cutting the dimension(s) that has biggest number of subvolumes degrades the performance.

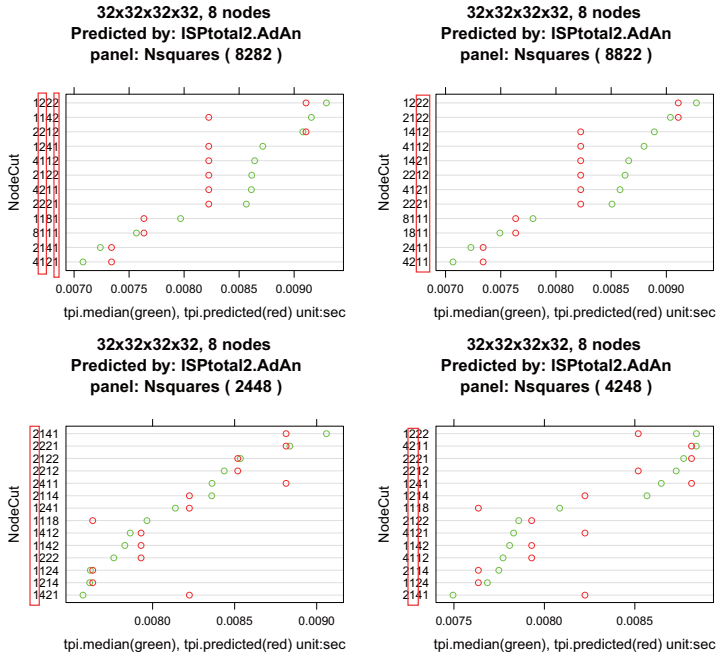

Figure 5.11: $32 \times 32 \times 32 \times 32$, 4 nodes. The tpi.median here can not be well predicted by ISPtotal2.AdAn. We can see certain pattern by examining the numbers in red box. Actually in this case tpi.median can be predicted by ISPmax2.1dAn (the max number of ISP of 4 dimensions of a node) and SITEStotal.1nAd (fig. 5.12).

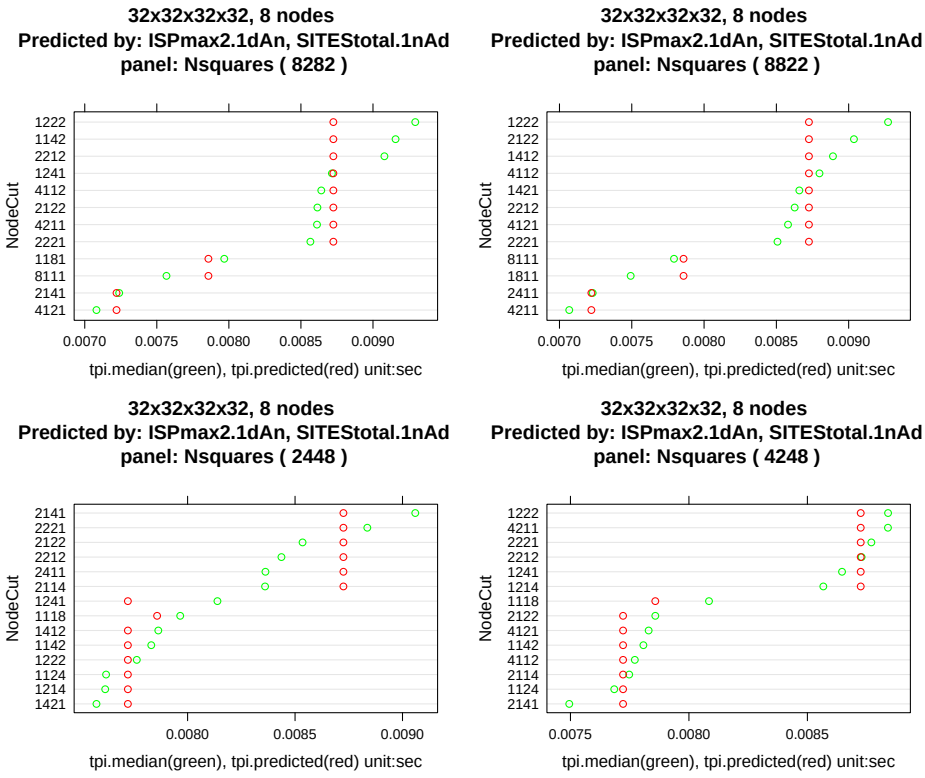

Figure 5.12: $32 \times 32 \times 32 \times 32$, 4 nodes.

Chapter 6

Number of Sites on inter-node surface

Number of Sites on inter-node surface has impact on the size of data trasferred over network. When the difference in number of sites on inter-node surface is big enought, it can have combined effects with ISPtotal2.AdAn.

When Nsquares is $4 \times 4 \times 4 \times 4$, the ISPtotal2.AdAn are same for different cuts, which means we can rule out the influences of ISPtotal2.AdAn in this case (Fig. 6.3). To double check this argument, we can check the performance on 24x12x12x12. Fig. 6.4

6.1 Experiments

To further confirm the change in ISPtotal2.AdAn and SITEStotal.1nAd, we can look at the communication heat map generated by TAU. We pick NodeCut 1214 and 1421 from Nsqares 4444 in lattice $12 \times 12 \times 12 \times 24$, 8 nodes. They are supposed to have same ISPtotal2.AdAn. NodeCut 1214 is supposed to have less SITEStotal.1nAd, which measn it has less bytes of data transferring over inter-node network.

Fig. 6.7 and 6.8 are the communication heat maps for NodeCut 1214 and 1421. Each node has 32 cores, so in the maps, rank 0-31 are in one node, rank 32-63 are in one node, ... We can observe that in each line of different NodeCut of the same heat map type, the numbers of different color points are the same. The difference is that they are arranged in a different way. The reason for this is that: a rank represents a subvolume, and the communication pattern for a subvolume and its adjacent subvolumes is fixed. By changing the layout, essentially we only change the ranks of its adjacent subvolumes. We can see that NodeCut 1214 and 1421 have same number of inter-node calls (we can ignore the blue points since each of them only represents 5 calls.), which is proportional to ISPtotal2.AdAn. We also can easily tell that NodeCut 1421 has more inter-node bytes since there are more inter-node red points.

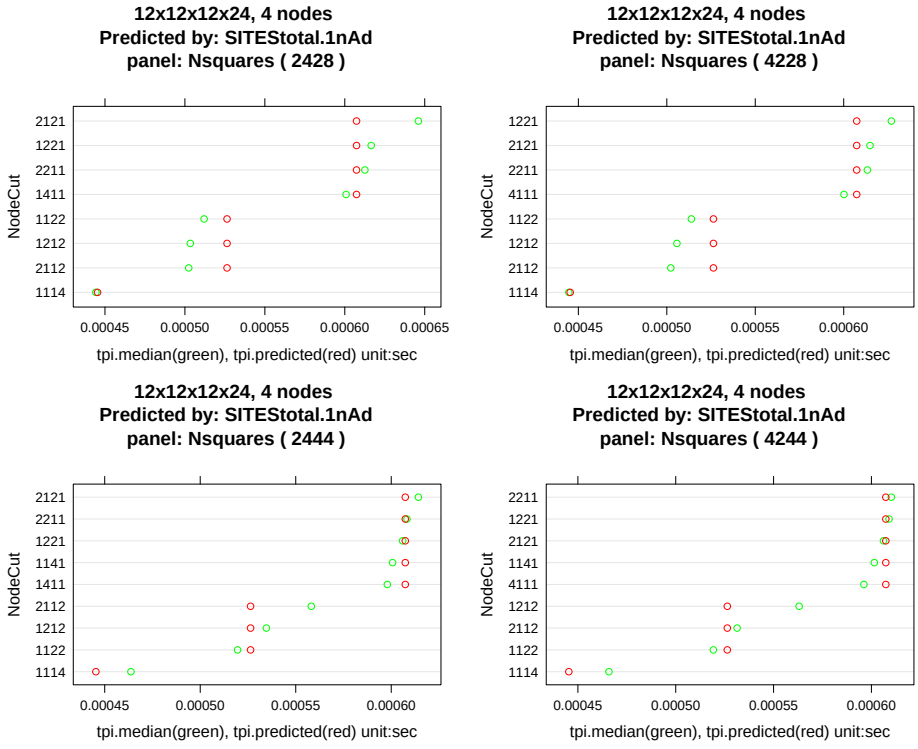

Figure 6.1: $12 \times 12 \times 12 \times 24$, 4 nodes. In this case, SITEStotal.1nAd is the dominant factor, so it can separate the performance to differnet groups.

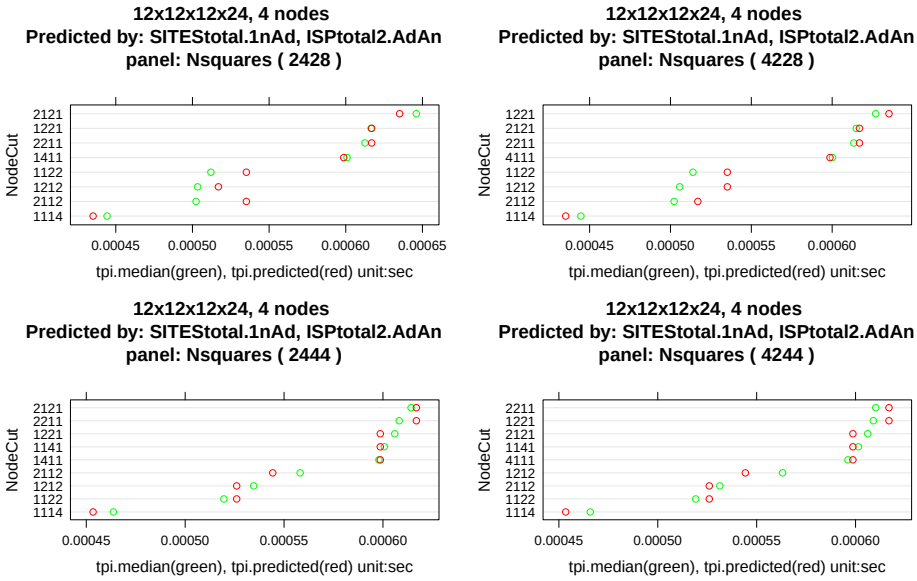

Figure 6.2: $12 \times 12 \times 12 \times 24$, 4 nodes. We can further categorize the performance by ISPtotal2.AdAn. We can see the performance matchs the prediction quite well.

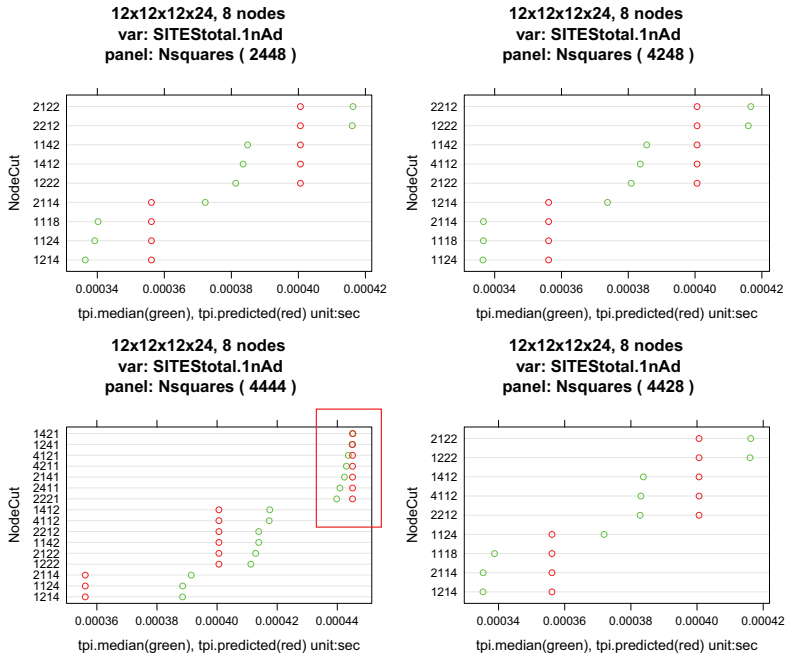

Figure 6.3: $12 \times 12 \times 12 \times 24$, 8 nodes.

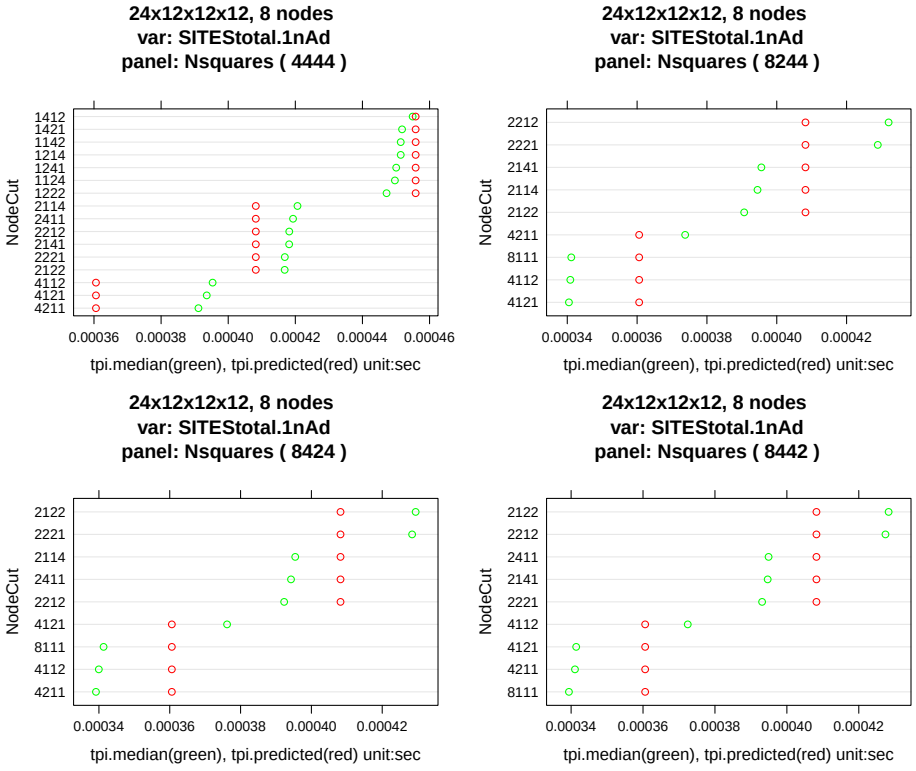

Figure 6.4: $24 \times 12 \times 12 \times 12$, 8 nodes.

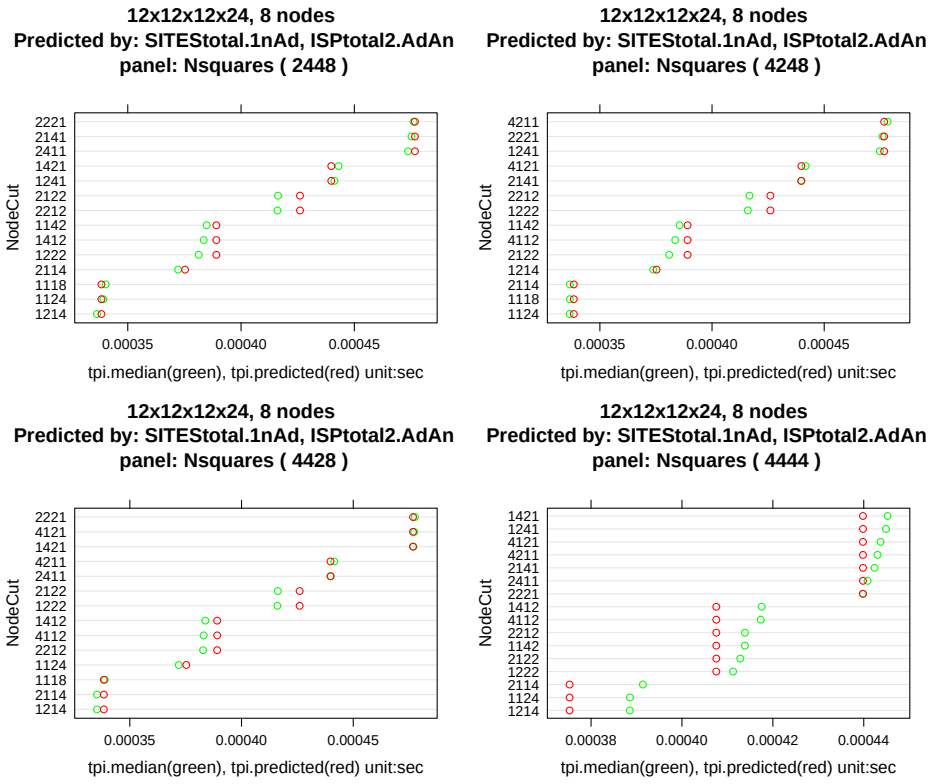

Figure 6.5: $12 \times 12 \times 12 \times 24$, 8 nodes. In this case, we can also see that the performance can first be categorized by SITEStotal.1nAd, and then be further categorized by ISPtotal2.AdAn. It matches the performance pattern.

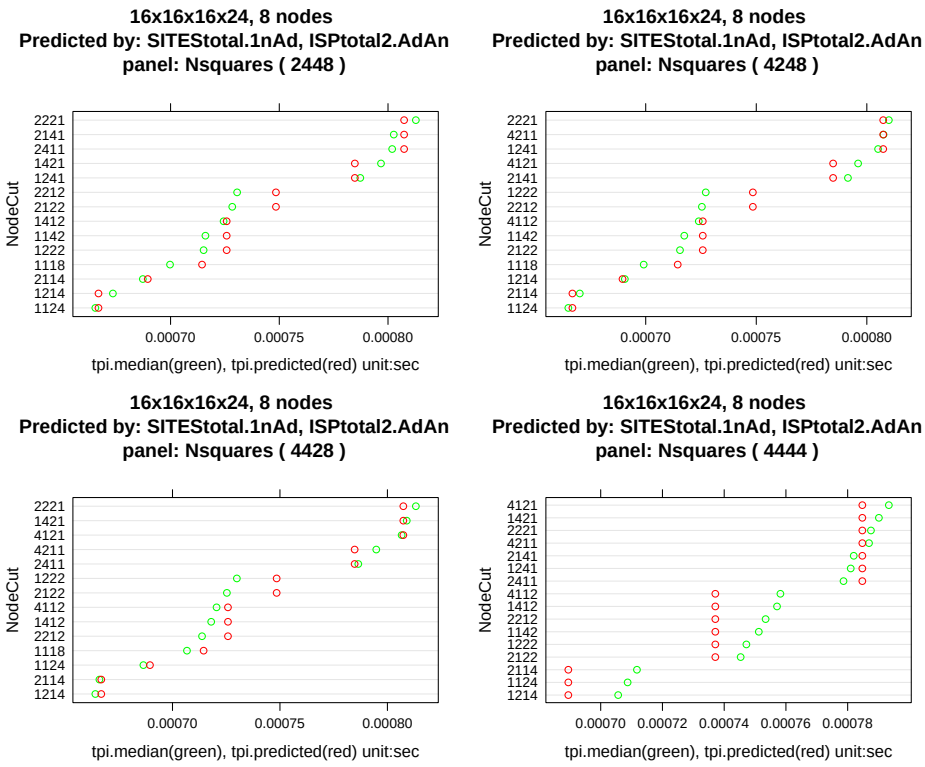

Figure 6.6: 16x16x16x24, 8 nodes. In this case, we can also see that the performance can first be categorized by SITEStotal.1nAd, and then be further categorized by ISPtotal2.AdAn. It matches the performance pattern.

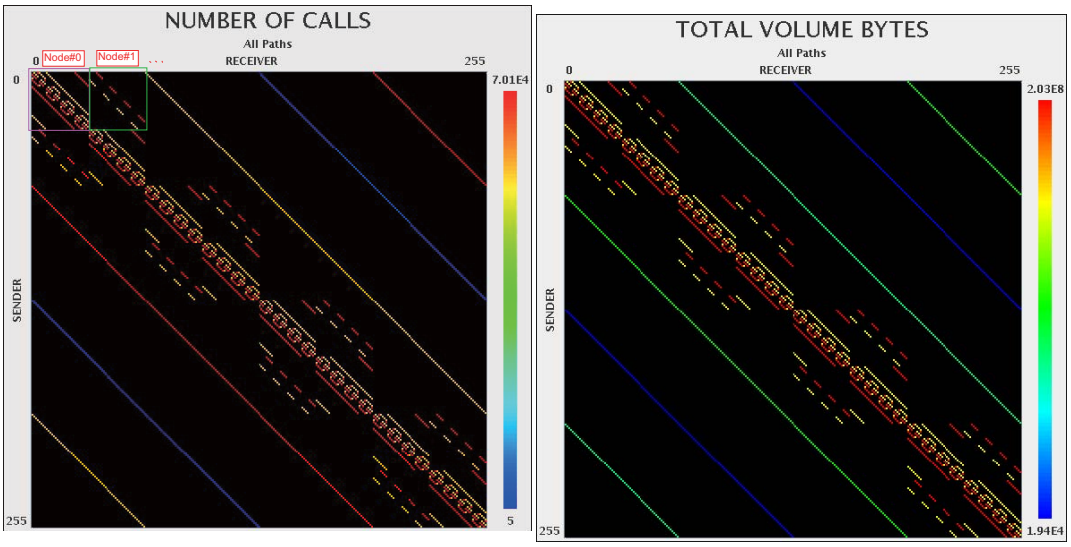

Figure 6.7: $12 \times 12 \times 12 \times 24$, 8 nodes, Nsquares: 4444, NodeCut: 1214

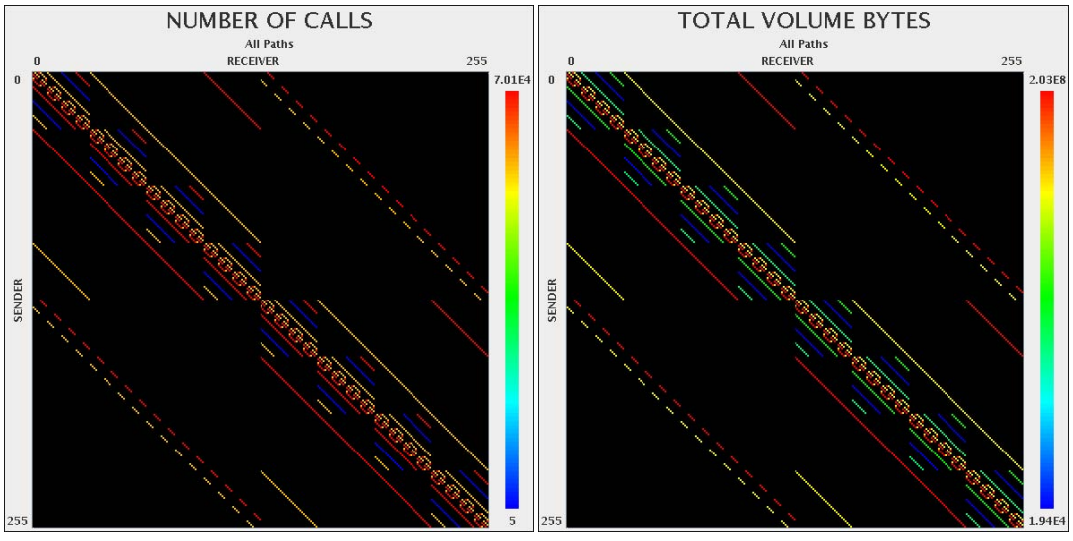

Figure 6.8: $12 \times 12 \times 12 \times 24$, 8 nodes, Nsquares: 4444, NodeCut: 1421

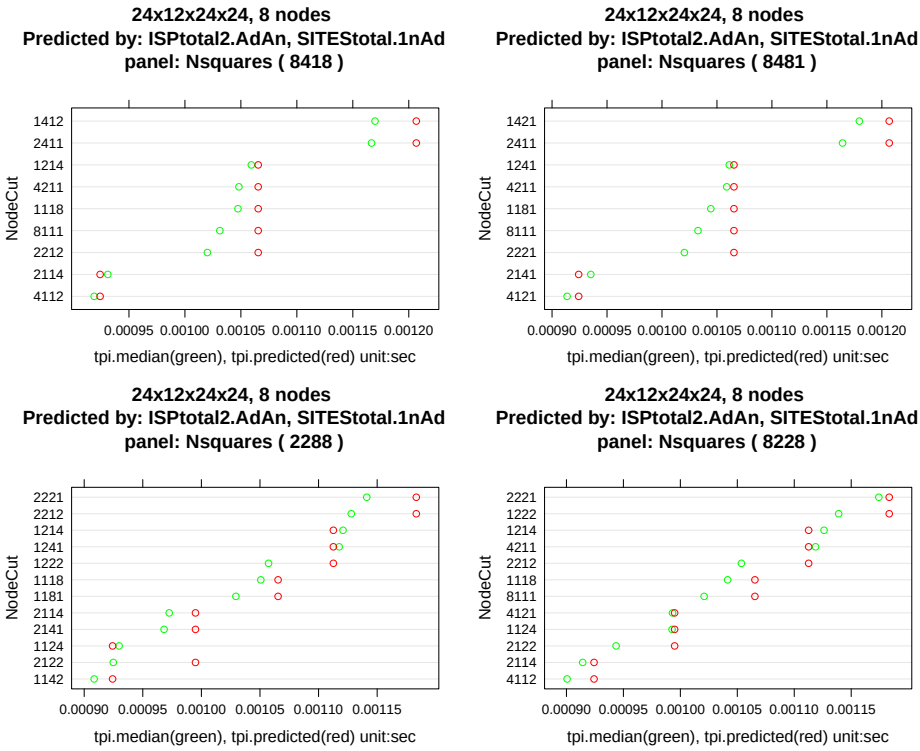

Figure 6.9: 24x12x24x24, 8 nodes. Well predicted by ISPtotal2.AdAn and SITEStotal.1nAd

Chapter 7

Max Number of Sites on inter-node surface per node

This variable can describe the max size of data transferred between two nodes. It is calculated by the function in Appendix .4.

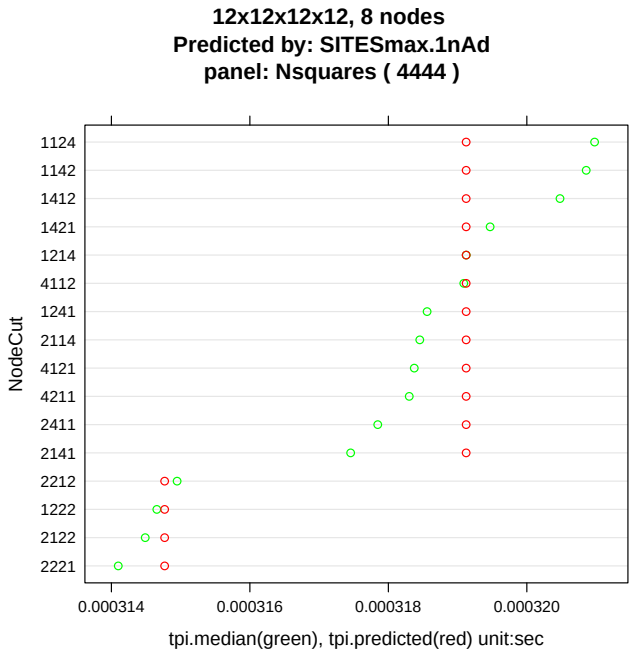

Figure 7.1: $12 \times 12 \times 12 \times 12$, 8 nodes. In this case, all the other variables are the same. So SITESmax.1nAd became the dominant. It brings only small difference in performance. Therefore it can easily be overshadowed by other variables. When the lattice size goes up to $16 \times 16 \times 16 \times 16$, it can not predict tpi.median.

Chapter 8

Intra-Node Cost

Intra-Node cost describes the communication cost within a node. This cost is introduced by the different distances between different cores. Fig. 8.1 is the topology of a D/S node. It has 32 cores, 8 NUMA nodes and 4 sockets. Below are the NUMA distances printed out by *numactl –hardware*.

node distances:
node 0 1 2 3 4 5 6 7
0: 10 16 16 22 16 22 16 22
1: 16 10 22 16 22 16 22 16
2: 16 22 10 16 16 22 16 22
3: 22 16 16 10 22 16 22 16
4: 16 22 16 22 10 16 16 22
5: 22 16 22 16 16 10 22 16
6: 16 22 16 22 16 22 10 16
7: 22 16 22 16 22 16 16 10

$intraSzCost = \sum_{i=1}^{numofpaths} size[i] \times distance[i]$. $instraSzCost$ is the intra-node cost. $size[i]$ is the size of data transferring on the ith path. $distance[i]$ is the distance of the ith path in this node.

8.1 Experiments

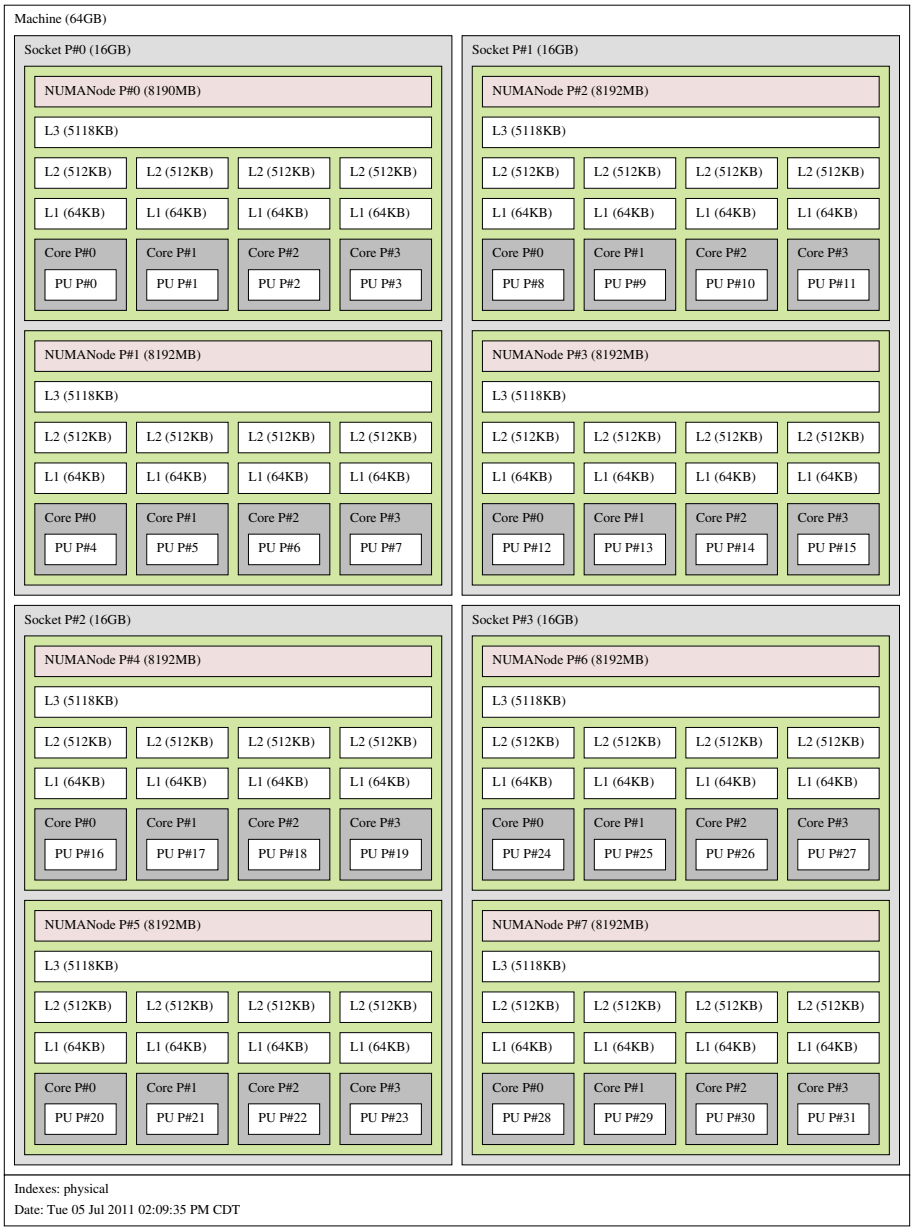

Figure 8.1: Node infrastructure generated by Hwloc.

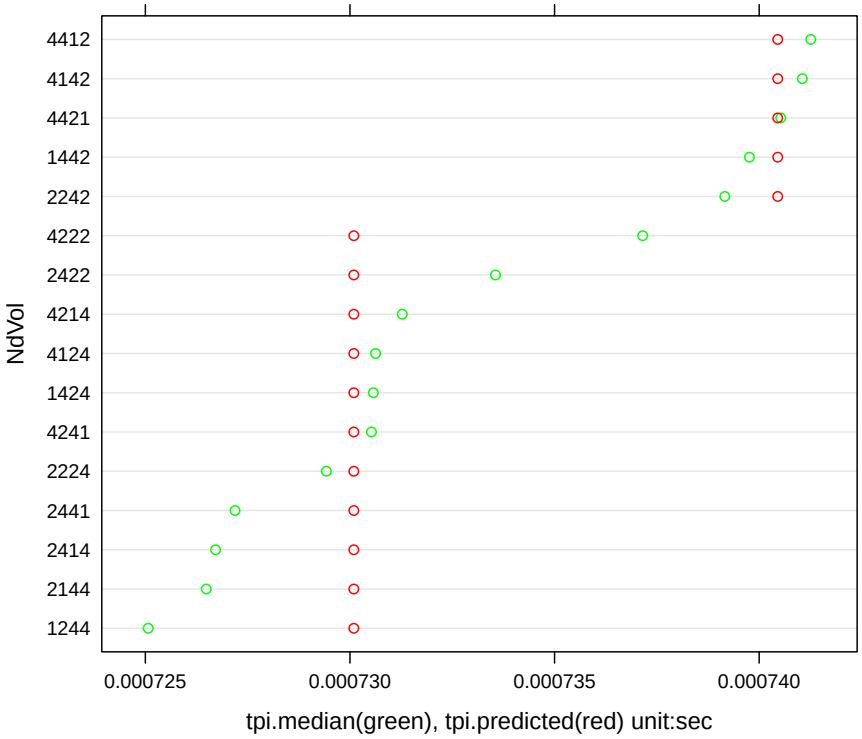

Figure 8.2: $12 \times 12 \times 12 \times 12$, 1 nodes. Because there is only one node here, all the subvolumes are in the same node and then $NdVol = Nsquares$. intraSzCost can predict tpi.median in this case. The range of tpi.median is $0.000740 - 0.000725 = 0.000015$. The max difference is 2%.

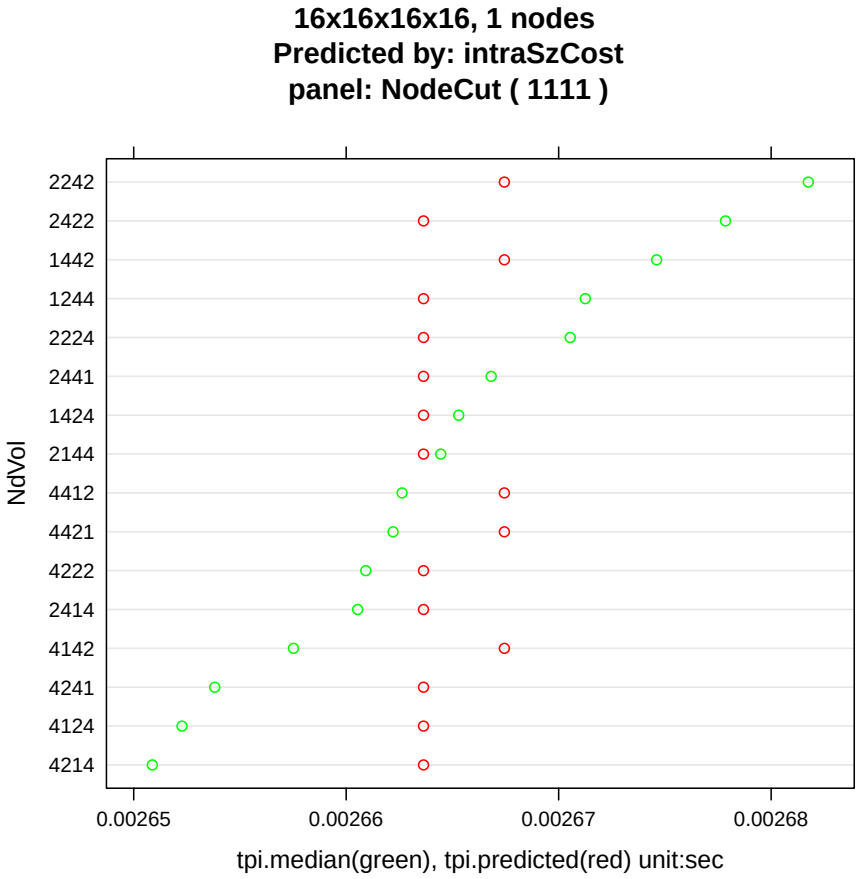

Figure 8.3: $16 \times 16 \times 16 \times 16$, 8 nodes. intraSzCost can no longer predict the tpi.median in this case, even though the size goes up a little. The range of tpi.median is $0.000268 - 0.000265 = 0.000003$. The max difference is 1.1%.

Chapter 9

Other Variables Inspected

There are many variables to be inspected. We have tried to use at most three of them to create the linear model and predict tpi.median. We only covered several most important ones in this report due to the limit of space and time. Here we give a brief introduction of each variable and their impacts on the performance. They are not as good as the variables mentioned above. The functions for calculating these variables are in the R scripts of this project. The function names are in the form of getVARIABLENAME(), for example, getISPmax.Ad1n(). The definitions below are vague, please refer to the R codes for details.

- ISPmax.Ad1n: the max number of inter-node paths of 4 dimensions for one node. For example, if the NodeVolume is $2 \times 4 \times 4 \times 1$, ISPmax.Ad1n is $2 \times 4 \times 4 = 32$, which indicates 32 paths in T dimension. (Assume that there is communication in T dimension.) tpi.median can not be predicted by ISPmax.Ad1n as well as ISPtotal2.AdAn, such as in the very typical case of lattice size $12 \times 12 \times 12 \times 24$ and 8 nodes.
- ISPmax3.AdSingleLine: this variable describes the max number of inter-node paths in all the nodes along that dimension. This variable can be used to predict tpi.median only when it happened to be proportional to other dominant variables.
- ISPmax4.1dAnAc: this is the max number of inter-node paths in all the nodes in that dimension. The difference between this and ISPmax3.AdSingleLine is that ISPmax4.1dAnAc includes all the nodes.
- ISPtotal.AdSingleLine: the total number of inter-node paths, one line of subvolumes in each dimension.
- ISPtotal.1n4d: total number of inter-node paths for one node.
- intraCost: the intra-node cost without taking into account the size of the transferred data.
- SITESmax.1nAd: max number of sites on one of four inter-node surfaces in one node.

- SITESmax.1nAdAc: max number of sites on one of four inter-node surfaces in all nodes.

Part V

Conclusion & Future Work

Chapter 10

Conclusion

By analyzing the performance, we found that the dominant factors include number of inter-node paths (ISPtotal2.AdAn), number of inter-node sites (SITEStotal.1nAd), max number of inter-node paths per node (ISPmax2.1dAn) and max number of inter-node sites per node (SITESmax.1nAd). The functions for calculating these variables are given in Appendix. They take dominant effect in different occasions.

When the lattice size is asymmetric, the number of sites on inter-node surface is the dominant factor. The group of layouts that has the least number of sites on the inter-node surfaces has the least TPI median. We can further find the best layout of all by number of inter-node paths. (fig. 6.5 and 6.6)

When the lattice size is small (number of sites per node) and symmetric, the TPI performance can be categorized by number of inter-node paths (fig. 5.1). For all the small-size cases we have tested, the sites on inter-node surfaces were all the same (because it requires large number of nodes to get small number of sites per node). My guess is that whenever there is a difference in number of sites per node (SITEStotal.1nAd), it will be the dominant factor.

When the lattice size is big (number of sites per node), the max number of inter-node paths becomes more important (fig. 5.12).

Chapter 11

Future Work

11.1 Layouts

- some more variables to look into: memory layout, the actual bandwidth of intra-node and inter-node communication.
- inspect the difference of cutting different dimensions.
- collect and analyze data in a bigger scale.

11.2 Congrad time goes up when the PBS job runs a long time.

In our experiments, we put hundreds of MPI runs in one PBS job. We discovered that the total congrad time went up for each MPI run (fig. 11.1). Future research can be done on this point.

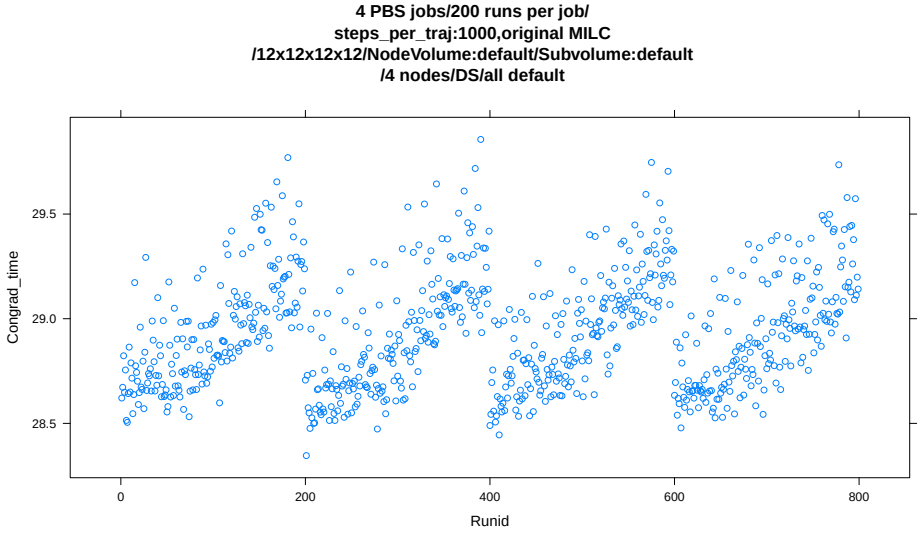

Figure 11.1: Congrad time goes up when the PBS job runs a long time. Starting a new job can reset the congrad time back to normal

Appendices

.1 ISPtotal2.AdAn

```
getISPtotal2.AdAn <- function( e ) {
  lt = cbind(e$LatticeX, e$LatticeY, e$LatticeZ, e$LatticeT)
  sv = cbind(e$Subvolume_X, e$Subvolume_Y, e$Subvolume_Z, e$Subvolume_T)
  nv = cbind(e$NodeVolume_X, e$NodeVolume_Y, e$NodeVolume_Z, e$NodeVolume_T)
  ns = lt/sv
  cut = lt/(sv*nv)

  nr = nrow(e)
  isptotal = mat.or.vec(nr,1)


  for ( i in 1:nr ) {
    # get total
    total = 0
    for ( j in 1:4 ) {
      if ( cut[i, j] > 1 ) {
        # dim j has been cut
        tmp = cut[i,j]*prod( ns[i, select=-j] )
        total = total + tmp
      }
    }
    isptotal[i] = total
  }
  return (isptotal)
}
```

.2 SITEStotal.1nAd

```
getSITEStotal.1nAd <- function (e) {
  lt = cbind(e$LatticeX, e$LatticeY, e$LatticeZ, e$LatticeT)
  sv = cbind(e$Subvolume_X, e$Subvolume_Y, e$Subvolume_Z, e$Subvolume_T)
  nv = cbind(e$NodeVolume_X, e$NodeVolume_Y, e$NodeVolume_Z, e$NodeVolume_T)
  ns = lt/sv
  cut = lt/(sv*nv)
  nv.s = nv*sv #size of subvol in sites

  nr = nrow(e)
  s.total = mat.or.vec(nr,1)

  #print(nr)
  for ( i in 1:nr ) {
    # get total
    total = 0
```

```
    #print(c(nv.s[i,],cut[i,]))
    for ( j in 1:4 ) {
      if ( cut[i, j] > 1 ) {
        # dim j has been cut

        tmp = 2*prod( nv.s[i, select=-j] )
        total = total + tmp
      }
    }
    s.total[i] = total
    #print(s.total[i])
  }
  return (s.total)
}
```

.3 ISPmax2.1dAn

```
getISPmax2.1dAn <- function( e ) {
  lt = cbind(e$LatticeX, e$LatticeY, e$LatticeZ, e$LatticeT)
  sv = cbind(e$Subvolume_X, e$Subvolume_Y, e$Subvolume_Z, e$Subvolume_T)
  nv = cbind(e$NodeVolume_X, e$NodeVolume_Y, e$NodeVolume_Z, e$NodeVolume_T)
  ns = lt/sv
  cut = lt/(sv*nv)

  nr = nrow(e)
  ispmax = mat.or.vec(nr,1)


  for ( i in 1:nr ) {
    # get max
    max = 0
    for ( j in 1:4 ) {
      if ( cut[i, j] > 1 ) {
        # dim j has been cut
        tmp = prod( ns[i, select=-j] )
        if ( tmp > max ) {
          max = tmp
        }
      }
    }
    ispmax[i] = max
  }
  return (ispmax)
}
```

.4 SITESmax.1nAd

```
getSITESmax.1nAd <- function( e ) {
  lt = cbind(e$LatticeX, e$LatticeY, e$LatticeZ, e$LatticeT)
  sv = cbind(e$Subvolume_X, e$Subvolume_Y, e$Subvolume_Z, e$Subvolume_T)
  nv = cbind(e$NodeVolume_X, e$NodeVolume_Y, e$NodeVolume_Z, e$NodeVolume_T)
  ns = lt/sv
  cut = lt/(sv*nv)
  nv.s = nv*sv #size of subvol in sites

  nr = nrow(e)
  s.max = mat.or.vec(nr,1)


  for ( i in 1:nr ) {
    # get max
    max = 0
    for ( j in 1:4 ) {
      if ( cut[i, j] > 1 ) {
        # dim j has been cut
        tmp = 2 * prod( nv.s[i, select=-j] )
        if ( tmp > max ) {
          max = tmp
        }
      }
    }
    s.max[i] = max
  }
  return (s.max)
}
```